

HIGH-PERFORMANCE PARALLEL INTERFACE - Scheduled Transfer (HIPPI-ST)

November 15, 1996

Secretariat:

Information Technology Industry Council (ITI)

ABSTRACT: This standard specifies a data transfer protocol that uses small control messages to pre-arrange data movement. Buffers are allocated at each end before the data transmission, allowing full-rate, non-congesting data flow between the end devices. The control and data may use different physical media, or may share a single physical medium. Procedures are provided for moving data over HIPPI and other media.

NOTE:

This is an internal working document of X3T11, a Technical Committee of Accredited Standards Committee X3. As such, this is not a completed standard. The contents are actively being modified by X3T11. This document is made available for review and comment only. For current information on the status of this document contact the individuals shown below:

POINTS OF CONTACT:

Roger Cummings (X3T11 Chairman)
Distributed Processing Technology
140 Candace Drive
Maitland, FL 32751
(407) 830-5522 x348, Fax: (407) 260-5366
E-mail: cummings_roger@dpt.com

Carl Zeitler (X3T11 Vice-Chairman)
IBM Corporation, MS 9440
11400 Burnet Road
Austin, TX 78758
(512) 838-1797, Fax: (512) 838-3822
E-mail: zeitler@ausvm6.vnet.ibm.com

Don Tolmie (HIPPI-ST Technical Editor)
Los Alamos National Laboratory
CIC-5, MS-B255
Los Alamos, NM 87545
(505) 667-5502, Fax: (505) 665-7793
E-mail: det@lanl.gov

Comments on Rev 0.1

This is a preliminary document undergoing lots of changes. Many of the additions are just place holders, or are put there to stimulate discussion. Hence, do not assume that the items herein are correct, or final – everything is subject to change. This page tries to outline where we are; what has been discussed and semi-approved, and what has been added or changed recently and deserves your special attention. This summary relates to changes since the previous revision. Also, previous open issues are outlined with a single box, new open issues ones are marked with a double bar on the left edge of the box.

Changes are marked with change bars in the margin. Minor changes, e.g., capitalization or spelling, did not warrant a change bar unless there were other substantive changes in the paragraph. The list below just describes the major changes, for detail changes please compare this revision to the previous revision.

Please help us in this development process by sending comments, corrections, and suggestions to the Technical Editor, Don Tolmie, of the Los Alamos National Laboratory, at det@lanl.gov. If you would like to address the whole group working on this document, send the STU to hippy@network.com.

1. Made major changes to the first paragraph of the Foreword, Introduction, and Scope. If you want to see what changed, it is best to compare to an old copy.
2. Global changes that are not marked with margin bars include: Changed "control Message" to "Control Operation". Changed "Message" to "STU", i.e., Scheduled Transfer Unit. Changed "M_count" to "S_count". Changed "operation" and "slot" to always begin with a capital letter.
3. In the Introduction, deleted some of the bullet items, and modified others.
4. Page 1, Scope – changed the bullet items that are marked.
5. 3.1.3, Changed the definition of "Control Channel" by deleting the words "e.g., Scheduled Headers".
6. 3.1.5, Changed the definition of "Final Destination" by changing "equipment" to "end device", and changed "...payload portion of the micropackets" to "...payload".
7. 3.1.7, changed the definition of Operation from "...e.g., Request_To_Send, defined in a 32-byte Scheduled Transfer Header." to "...i.e., a Control Operation or the data movement specified in an STU.".
8. 3.1.9, changed the definition of Originating Source from "...equipment..." to "...end device..." and from "...payload portion of the micropackets." to "...payload.".
9. 3.1.12, added the definition for Scheduled Transfer Unit.
10. 3.1.15, changed the definition of Virtual Connection from "...full-duplex..." to "...bi-directional...".
11. In 3.3, added acronyms for MPI and STU.
12. In 4.1, deleted the last sentence of the first paragraph about the control and data channels using different or same media.
13. In almost a global change, changed "...32-byte Schedule Header..." to "...Schedule Header...". Changed "Scheduled Header" to "Schedule Header" throughout the document. Changed "user data", and other similar variants to "data payload".
14. In figure 1, changed the titles from "In end device.." to "Device...".
15. The first paragraph of 4.2 was mostly lifted from 4.3 of the previous revision. The last two sentences were added to explain the new items in figure 2.
15. Added a picture of an STU and Control Operation to figure 2.
16. The rest of 4.2 is all new, as well as figure 3. This is an attempt to better define what we are talking about.
17. 4.3 is text that was previously 4.2; with changes. In the first paragraph, the next to last sentence was changed from "A path..." to "A bi-directional path..." and from "...Originating Source and Final Destination." to "...end devices.". In the second paragraph, 1st sentence, changed "Since..." to "Once...", changed in the next to last sentence from "...move multiple..." to "...carry multiple...". Added the last sentence about this protocol doesn't handle network resource reservations.

18. 4.3.1, 1st paragraph, the last sentence was added to reference where in the model we are describing. In the 2nd paragraph, replaced the second sentence "Once established, the Virtual Connection is accessed as shown in figure 3 by the tuple "his Port", "my Port", and "my Key".".
19. 4.3.2, changed the last words in the paragraph from "...is ignored." to "...shall not be executed.".
20. 4.3.3, changed the 1st sentence from "Ports are logical..." to "Ports identify higher-layer logical...". Changed the 3rd sentence from "...a well-known port, e.g., a port for IP traffic." to "...the well-known port, i.e., port x'0000'.".
21. 4.3.4, changed the first sentence from "...defines the maximum buffer size..." to "...shall define the buffer size...". Changed the 2nd sentence from "...smaller of these shall be the buffer size..." to "...smaller of these shall be the maximum Message size...".
22. 4.3.5, changed the first sentence to note that the Slots are per port, and calling it a Schedule Header queue rather than an Operation queue. Added the sentence about reserving an extra slot for End. Changed the names of the operations that request and supply the state.
23. 4.4.1, added "(T_id's)". Deleted that the Control Operations use 32-byte Schedule Headers.
24. 4.4.1, 2nd paragraph, deleted the 32-bytes + ?? for Schedule Header size. Added "...e.g., with a Request_To_Send". Reordered the sentences in the paragraph for easier comprehension.
25. 4.4.1, 3rd paragraph, deleted the 32-bytes for the Schedule Header.
26. 4.4.1, 4th paragraph, changed the first sentence from "Messages" to "STUs"; from "32-byte Schedule Headers followed by user data" to "40-byte Schedule Headers followed by data payload". The second and third sentences of the original paragraph were removed. In the last sentence, changed "...if an acknowledgment should be sent" to "...if state information should be sent.".
27. 4.4.1, 5th paragraph, changed acknowledging to acquiring state. Changed the name of the operations from DATA_ACK to State_Response, and from Request_ACK to State_Request.
28. In 4.4.4, changed "Message" to "STU" throughout. Changed the last sentence from "...shall extend past a..." to "...shall extend past a Final Destination's...".
29. In 6, added the 8 bytes of mandatory payload at the end of figure 5. Added the text at the end of this section.
30. In 6.2, reordered the flag bits so that they were in more logical groupings, i.e., like bits together. Added the Interrupt bit. Changed the Notify bit from "...notify..." to "...deliver this STU's Schedule Header to...".
31. In 6.2, and throughout the document, changed the "DATA_ACK_Requested" flag to the "State_Requested" flag. Added the sentence "For State_Requested to be valid, either the Interrupt or Notify flag must also =1.".
32. In 7, 4th paragraph, changed the don't care value from x'0000' to x'FFFFFFF'. Added that one extra Slot should be reserved for an End operation.
33. In 7.1, 4th paragraph, changed the A-limit text to match what is now in clause 7.
34. In 7.1, 5th paragraph, added text about the well-known port, and decoding on EtherType when using it.
35. In 7.2, 4th paragraph, changed the B-limit text to match what is now in clause 7, and 7.1.
36. In 7.3, 7.4, and 7.5, this text is still under discussion and further changes may be made. The expected changes involve text or figures describing the 3-way handshake.
37. In 8.5, in the next to last sentence, added "e.g., through a previous RTS Operation".
38. In 8.6, the flags parameter was expanded to include the Interrupt flag, and the text was reordered to match the order of flags in the flag field. The last sentence was removed from the Sync parameter giving an algorithm for the Sync value.
39. In 8.7 and 8.8, the "DATA_ACK_Request" operation was renamed "State_Response", and the "Request_ACK" operation was renamed "State_Request". The order of the operations were reversed in the text, and the op codes were also swapped, i.e.,

Request comes before Response. The all ones value for D_id and S_id was added. Added the B_seq parameter in 8.8.

40. Tables 1 and 2 were updated to match the changes to the rest of the text.
41. No changes were made to Annex C other than the global name changes.

Contents

| | Page |
|---|------|
| Foreword..... | vii |
| Introduction..... | viii |
| 1 Scope..... | 1 |
| 2 Normative references..... | 1 |
| 3 Definitions and conventions..... | 2 |
| 3.1 Definitions..... | 2 |
| 3.2 Editorial conventions..... | 2 |
| 3.3 Acronyms and other abbreviations..... | 3 |
| 4 System overview..... | 3 |
| 4.1 Control Channels and Data Channels..... | 3 |
| 4.2 System model..... | 4 |
| 4.3 Virtual Connections..... | 6 |
| 4.4 Data movement..... | 7 |
| 5 Service interface..... | 9 |
| 5.1 Service primitives..... | 9 |
| 5.2 Sequences of primitives..... | 9 |
| 6 Schedule Header..... | 10 |
| 6.1 Schedule Header parameters..... | 10 |
| 6.2 Scheduled Transfer flags..... | 10 |
| 7 Virtual Connection Operations..... | 11 |
| 7.1 Request_Port (RQP)..... | 12 |
| 7.2 RQP_Response..... | 12 |
| 7.3 Port_Tearardown..... | 13 |
| 7.4 Port_Tearardown_ACK..... | 13 |
| 7.5 Port_Tearardown_Complete..... | 14 |
| 8 Scheduled Transfer data Operations..... | 14 |
| 8.1 Request_To_Send (RTS)..... | 14 |
| 8.2 RTS_Response..... | 15 |
| 8.3 Clear_To_Send (CTS)..... | 15 |
| 8.4 RTS_Response/CTS..... | 16 |
| 8.5 Request_To_Receive (RTR)..... | 16 |
| 8.6 DATA..... | 17 |
| 8.7 State_Request..... | 18 |
| 8.8 State_Response..... | 18 |
| 8.9 End..... | 19 |
| 8.10 End_ACK..... | 19 |
| 9 Sender detected errors..... | 19 |
| 10. Receiver detected errors..... | 21 |
| 10.1 General errors..... | 21 |
| 10.2 Virtual Connection errors..... | 21 |
| 10.3 Scheduled Transfer errors..... | 21 |
| 10.4 Other errors..... | 22 |

Tables

| | |
|--|----|
| Table 1 – Virtual Connection Operations summary..... | 20 |
| Table 2 – Data transfer Operations summary..... | 20 |
| Table 3 – Summary of logged errors..... | 22 |
| Table C.1 – Scheduled Transfer example summary..... | 29 |

Figures

| | |
|--|----|
| Figure 1 – System overview..... | 3 |
| Figure 2 – Information hierarchy..... | 4 |
| Figure 3 – Scheduled Transfer Final Destination model..... | 5 |
| Figure 4 – HIPPI-ST service interface..... | 9 |
| Figure 5 – Schedule Header contents..... | 10 |
| Figure C.1 – Buffer tiling..... | 27 |
| Figure C.2 – Alternate buffer tiling..... | 27 |

Annexes

| | |
|---|----|
| A Using HIPPI-6400-PH as the lower layer..... | 23 |
| B Using HIPPI-FP as the lower layer..... | 24 |
| C Scheduled Transfer example..... | 25 |
| C.1 Virtual Connection setup..... | 25 |
| C.2 Scheduled Transfer setup..... | 25 |
| C.4 Block 1 CTS..... | 27 |
| C.5 Block 246 transfer..... | 27 |
| C.6 Ending the Virtual Connection..... | 28 |

Foreword (This foreword is not part of American National Standard X3.xxx-199x.)

This American National Standard specifies a data transfer protocol that uses small control messages to pre-arrange data movement. Buffers are allocated at each end before the data transmission, allowing full-rate, non-congesting data flow between the end devices. The control and data may use different physical media, or may share a single physical medium. Procedures are provided for moving data over HIPPI and other media.

This standard provides an upward growth path for legacy HIPPI-based systems.

This document includes annexes which are informative and are not considered part of the standard.

Requests for interpretation, suggestions for improvement or addenda, or defect reports are welcome. They should be sent to the X3 Secretariat, Information Technology Industry Council, 1250 Eye Street, NW, Suite 200, Washington, DC 20005.

This standard was processed and approved for submittal to ANSI by Accredited Standards Committee on Information Processing Systems, X3. Committee approval of the standard does not necessarily imply that all committee members voted for approval. At the time it approved this standard, the X3 Committee had the following members:

(List of X3 Committee members to be included in the published standard by the ANSI Editor.)

Subcommittee X3T11 on Device Level Interfaces, which developed this standard, had the following participants:

(List of X3T11 Committee members, and other active participants, at the time the document is forwarded for public review, will be included by the Technical Editor.)

Introduction

This American National Standard specifies a data transfer protocol that uses small control messages to pre-arrange data movement. Buffers are allocated at each end before the data transmission, allowing full-rate, non-congesting data flow between the end devices. The control and data may use different physical media, or may share a single physical medium. Procedures are provided for moving data over HIPPI and other media.

Characteristics of a HIPPI-ST include:

- A hierarchy of data units (Scheduled Transfer Units (STUs), Blocks, and Transfers).
- Support for Get and Put Operations.
- Parameters exchanged between end devices for port selection, transfer identification, and Operation validation.
- Features supporting efficient mapping between the sender's and receiver's natural buffer sizes.
- Provisions for resending partial Transfers for error recovery.
- Mappings onto HIPPI-6400-PH, HIPPI-FP (for HIPPI-800 traffic), and Gigabit Ethernet lower-layer protocols.
- Mappings from IP, IPv6, and MPI upper-layer protocols onto Scheduled Transfer.

American National Standard for Information Technology –

High-Performance Parallel Interface – Scheduled Transfer (HIPPI-ST)

1 Scope

This American National Standard specifies a data transfer protocol that uses small control messages to pre-arrange data movement. Buffers are allocated at each end before the data transmission, allowing full-rate, non-congesting data flow between the end devices. The control and data may use different physical media, or may share a single physical medium. Procedures are provided for moving data over HIPPI and other media.

Specifications are included for:

- Virtual Connection setup and teardown;
- determining the number of Operations the other end can accept;
- determining the buffer size of the other end;
- exchanging Key, Port, and transfer identifiers, and buffer size values, specific to the end nodes;
- determining a maximum size that will not overrun receiver buffer boundaries;
- using buffer indices and 64-bit addresses;
- acknowledging partial transfers so that buffers can be reused;
- providing means for resending partial Transfers for error recovery; and
- terminating transfers in progress.

2 Normative references

The following American National Standards contain provisions which, through reference in this text, constitute provisions of this American National Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below.

ANSI X3.183-1991, *High-Performance Parallel Interface – Mechanical, Electrical, and Signalling Protocol Specification (HIPPI-PH)*

ANSI X3.210-1992, *High-Performance Parallel Interface – Framing Protocol (HIPPI-FP)*

ANSI X3.xxx-199x, *High-Performance Parallel Interface – 6400 Mbit/s Physical Layer (HIPPI-6400-PH)*

ANSI/IEEE Std 802-1990, *IEEE Standards for Local and Metropolitan Area Networks: Overview and architecture (formerly known as IEEE Std 802.1A, Project 802: Local and Metropolitan Area Network Standard — Overview and Architecture).*

ISO/IEC 8802-2:1989 (ANSI/IEEE Std 802.2-1989), *Information Processing Systems – Local Area Networks – Part 2: Logical link control.*

3 Definitions and conventions

3.1 Definitions

For the purposes of this standard, the following definitions apply.

3.1.1 Block: An ordered set of one or more STUs within a Scheduled Transfer. (See 4.3.3.)

3.1.2 Concatenate: An addressing mode using 64-bit addresses rather than buffer indices. (See 6.2.)

3.1.3 Control Channel: The logical channel that carries the Control Operations.

3.1.4 Data Channel: The logical channel that carries the data payload.

3.1.5 Final Destination: The end device that receives, and operates on, the data payload. This is typically a host computer system, but may also be a translator, bridge, or router.

3.1.6 HIPPI port: A HIPPI-6400-PH, or HIPPI-PH, Source or Destination.

3.1.7 Operation: A Scheduled Transfer function, i.e., a Control Operation or the data movement specified in an STU.

3.1.8 optional: Characteristics that are not required by HIPPI-ST. However, if any optional characteristic is implemented, it shall be implemented as defined in HIPPI-ST.

3.1.9 Originating Source: The end device that generates the data payload. This is typically a host computer system, but may also be a translator, bridge, or router.

3.1.10 Persistent: A control mode used to retain buffers for multiple Transfers. (See 6.2.)

3.1.11 Scheduled Transfer: An information transfer, normally used for bulk data movement and low processing overhead, where the Originating Source and Final Destination prearrange the transfer using the protocol defined in this standard.

3.1.12 Scheduled Transfer Unit (STU): The unit consisting of the Schedule Header and data payload moved from an Originating Source to a Final Destination. STUs are the basic components of Blocks.

3.1.13 Slot: A space reserved for a Control

Operation, or the Schedule Header portion of an STU, in the end device.

Open Issue – Do STUs without Notify or Interrupt = 1 need a Slot?

3.1.14 Transfer: An ordered set of one or more Blocks within a Scheduled Transfer. (See 4.4.)

3.1.15 Virtual Connection: A bi-directional logical connection used for Scheduled Transfers between an Originating Source and a Final Destination. A Virtual Connection contains a logical Control Channel and a logical Data Channel in each direction.

3.2 Editorial conventions

In this standard, certain terms that are proper names of signals or similar terms are printed in uppercase to avoid possible confusion with other uses of the same words (e.g., DATA). Any lowercase uses of these words have the normal technical English meaning.

A number of conditions, sequence parameters, events, states, or similar terms are printed with the first letter of each word in uppercase and the rest lowercase (e.g., Block, Transfer). Any lowercase uses of these words have the normal technical English meaning.

The word *shall* when used in this American National standard, states a mandatory rule or requirement. The word *should* when used in this standard, states a recommendation.

3.2.1 Binary notation

Binary notation is used to represent relatively short fields. For example a two-bit field containing the binary value of 10 is shown in binary format as b'10'.

3.2.2 Hexadecimal notation

Hexadecimal notation is used to represent some fields. For example a two-byte field containing a binary value of b'11000100 00000011' is shown in hexadecimal format as x'C403'.

3.3 Acronyms and other abbreviations

| | |
|--------------|-------------------------------------|
| ACK | acknowledge indication |
| CTS | Clear_To_Send |
| HIPPI | High-Performance Parallel Interface |
| lsb | least significant bit |
| MAC | Media Access Control |
| MPI | Message Passing Interface |
| msb | most significant bit |
| RQP | Request_Port |
| RTR | Request_To_Receive |
| RTS | Request_To_Send |
| STU | Scheduled Transfer Unit |
| ULP | upper-layer protocol |

4 System overview

This clause provides an overview of the structure, concepts, and mechanisms used in Scheduled Transfers. Figure 1 gives an example of Scheduled Transfers being used to communicate between device A and device B over some physical media. Annex C describes the steps in a typical Scheduled Transfer.

4.1 Control Channels and Data Channels

Each Scheduled Transfer instance has an Originating Source and Final Destination. Each Originating Source and Final Destination shall have a Control Channel, and one or more Data Channels.

Control Operations shall be exchanged over the Control Channel, Scheduled Transfer Units (STUs), i.e., data, shall be exchanged over the Data Channel(s). Control Operations shall consist of Schedule Headers. STUs shall consist of a Schedule Header and up to 4 gigabytes of data payload. The information volume on the Data Channel(s) will probably be many times the volume on the Control Channel, hence the available bandwidths should be balanced accordingly. For best performance, the Control Channel should have low latency, and should be separate from the Data Channel(s).

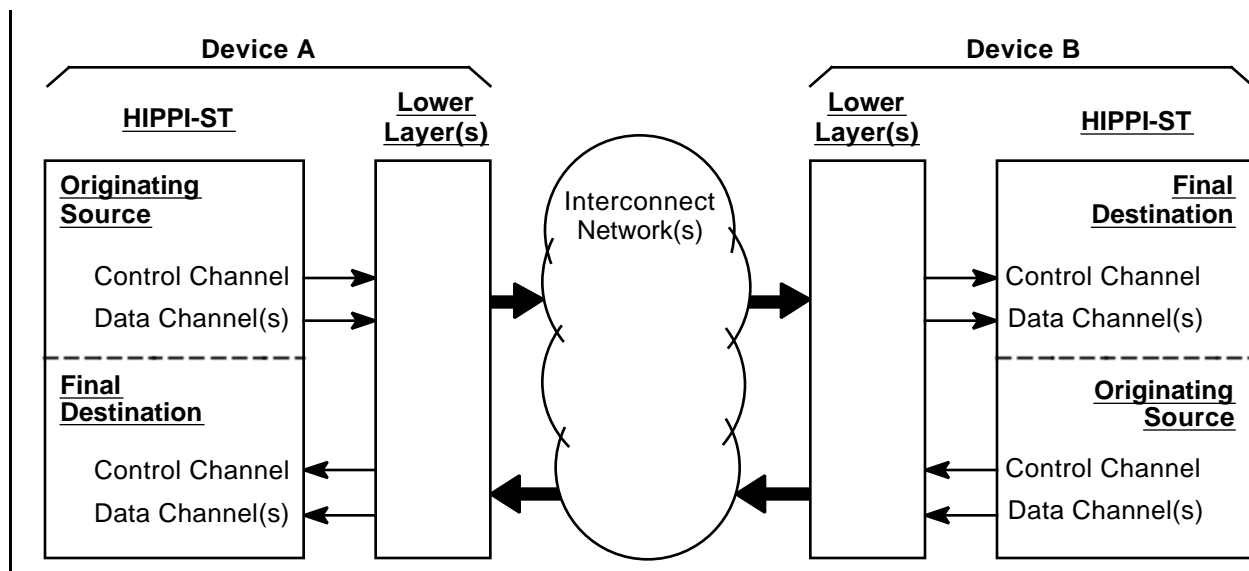
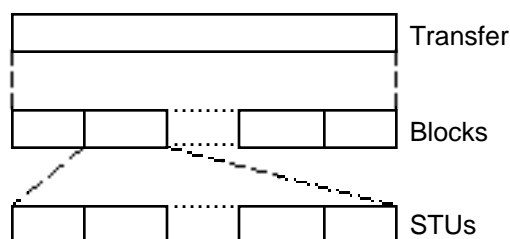


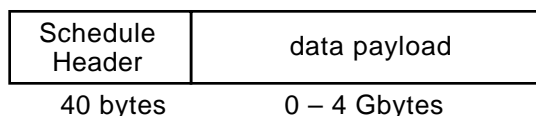
Figure 1 – System overview

4.2 System model

Multiple write (Put) or read (Get) functions may be executed to move data units, called Transfers, over a Virtual Connection. As shown in figure 2, a Transfer is composed of one or more Blocks, and Blocks are composed of one or more STUs. This Scheduled Transfer protocol shall package the Transfer in Blocks and STUs for delivery using lower layer protocol(s) and media. The STUs shall consist of a 40-byte Schedule Header followed by data payload. Control Operations shall consist of a 40-byte Schedule Header, and may contain up to 32 bytes of optional payload.



Scheduled Transfer Unit (STU)



Control Operation

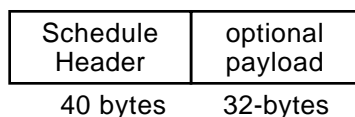


Figure 2 – Information hierarchy

Figure 3 shows the model used on a Final Destination for the Scheduled Transfers. The model on the Originating would probably be similar.

As Control Operations and STUs containing data payload are received, the Schedule Header of each is placed in the Schedule Header queue for execution. State information about the number of empty Slots in the queue is available to the other end so that it can avoid overrunning the queue.

The Virtual Connection Descriptor contains:

- static parameters defining the Virtual Connection from the view of both the remote end and local end (the top portion of the Virtual Connection Descriptor box in figure 3),
- current state information about the number of empty "Slots" for Schedule Headers, and
- identifiers for each of the Virtual Connection's Transfers.

A Transfer Descriptor, for each Transfer, contains the data size, in bytes, and includes pointers to Block Descriptors. The Block Descriptors (one for each Block of a Transfer) contain pointers to the Bufx and Offset tables for each STU of a Block. And finally, the STU Buffer Pointers are tables pointing to the buffer locations in the Final Destination's memory.

In an effort to achieve maximum transfer rates and efficiency, the receiver's job is made as easy as possible, even at the expense of the transmit side. It is expected that after validating an Operation in the Final Destination, only a single lookup will be needed to derive the absolute memory address and correctly place the data.

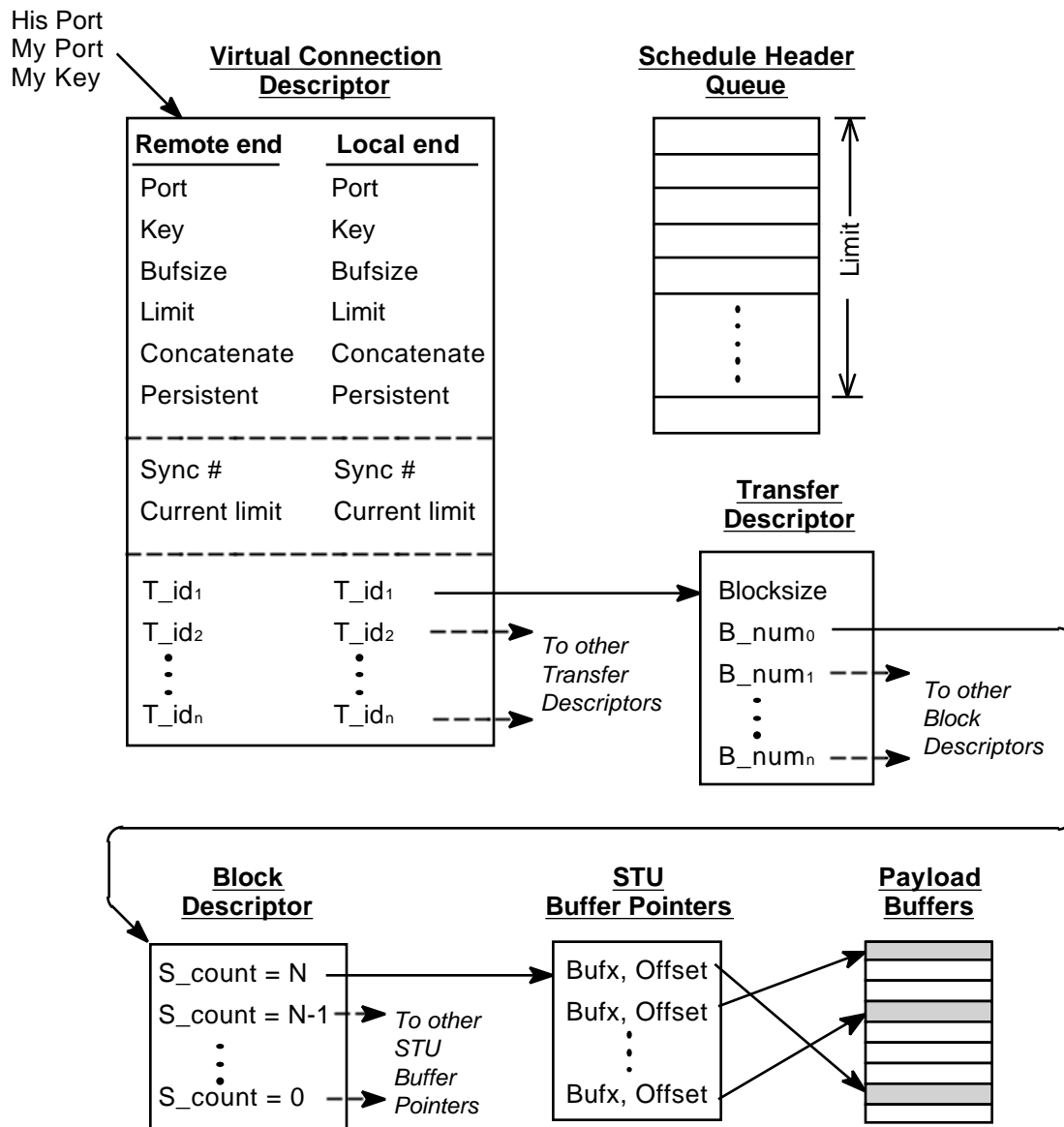


Figure 3 – Scheduled Transfer Final Destination model

4.3 Virtual Connections

Scheduled Transfers between an Originating Source and Final Destination are pre-arranged so that the transfer can occur at full rate. Pre-arrangement means that buffers are allocated at each end device. The bi-directional path between the end devices is called a Virtual Connection. A Virtual Connection shall consist of an Originating Source and Final Destination in each end device.

Once the Final Destination has indicated its ability to accept the STUs, the Virtual Connection should not become congested. In essence, the Final Destination smoothly controls the flow. For comparison, without pre-arranging the buffers the Originating Source blindly sends STUs into the interconnection network where they may have to wait for buffers to be assigned in the Final Destination. On the down-side, Scheduled Transfers require additional Control Operations, and round-trip latency. Once established, a Virtual Connection may be used to carry multiple Transfers. This Scheduled Transfer protocol does not handle network resource reservations.

4.3.1 Sequences and Operations

During Virtual Connection setup, the end devices shall exchange parameters specific to each device. These parameters, shown in the upper portion of the Virtual Connection descriptor box in figure 3 and detailed below, include values for:

- keys (used for authenticating Operations);
- Port numbers (e.g., a port for IP traffic);
- native buffer sizes (for determining STU sizes);
- maximum number of outstanding Operations (to keep from overflowing the command queues); and
- whether or not they support Concatenate and Persistent modes.

The parameters assigned during setup shall apply for the life of the Virtual Connection. Once established, the Virtual Connection is accessed as shown in figure 3 by the tuple "his Port", "my Port", and "my Key". The

Control Operations defined for Virtual Connection setup are:

- Request_Port (RQP) (See 7.1.)
- RQP_Response (See 7.2.)

The Control Operations defined for Virtual Connection teardown are:

- Port_Teardown (See 7.3.)
- Port_Teardown_ACK (See 7.4.)
- Port_Teardown_Complete (See 7.5.)

4.3.2 Keys

Each end device shall select its own 32-bit key value for use on the Virtual Connection. For example, when device A requests a Virtual Connection to device B, device A shall select the value for A-Key, and shall send it to device B in the Request_Port Operation. Device B shall store the A-Key value, and shall return it to device A in every Operation over this Virtual Connection. The A-Key value has no meaning in device B, it is only significant in device A where it shall be used to validate that the Operation presented is really associated with this Virtual Connection. Likewise, device B shall select the value for B-Key. Keys are similar in nature to passwords; if the key doesn't match, then the Operation shall not be executed.

4.3.3 Ports

Ports identify higher-layer logical connections within a device. Like the keys, the port values shall be assigned by the local device, and have no meaning on the other end. An exception is the "well-known port", i.e., port x'0000'. In this case, a request sent to the "well-known port" shall result in the receiving device assigning a specific local port value now that it knows the traffic to expect.

4.3.4 Buffer sizes

Each end shall define the buffer size, in bytes, that it wants to use. The smaller of these shall be the maximum Message size used on this Virtual Connection. Buffer sizes may be the same as host page sizes. It is most efficient when the buffer sizes are the same on both ends, but differing buffer sizes are supported

(see annex C). The buffer sizes shall be ≥ 32 bytes and shall be an integral power of two. The buffer index (Bufx) parameter can be used as an index into tables pointing to the buffers.

4.3.5 Operation limits

The current limit parameter defines the maximum number of Schedule Headers that a port is able to accept (one per Slot) without overflowing its Schedule Header queue. The number of Slots available is passed to the other end, where it is used to control the issuance of Control Operations and STUs, i.e., don't overrun or excess entities may be discarded. The device shall advertise at least one less than the actual number of Slots so that a Slot for an End is always available. A snapshot of the available Slots shall be returned with State_Response Control Operations (see 8.8), and can be requested with State_Request Control Operations (see 8.7).

Open Issue – The text beyond this point has not been reviewed in detail.

4.3.6 Concatenate and Persistent

Concatenate is a mode where the addressing method is changed (see 6.2). Persistent is a request to retain the buffers for multiple transfers rather than releasing them at the end of the transfer (see 6.2). Both Concatenate and Persistent are only usable between hosts that mutually agree. Agreement is reached by passing the associated flag bits during the Virtual Connection setup.

4.4 Data movement

4.4.1 Sequences and Operations

A write data sequence (which may be initiated by either end of the Virtual Connection) shall be set up by the end devices exchanging transfer identifiers (T_id's) specific to each device. Other parameters exchanged shall be the number of bytes in the Transfer, and the number of Blocks in the Transfer. The Control Operations setting up a write data sequence are:

- Request_To_Send (RTS) (See 8.1.)
- RTS_Response (See 8.2.)

A read data sequence, which moves the Transfer as a single Block, requires that both ends had previously allocated resources for the entire read sequence, e.g., with a Request_To_Send. The Control Operation setting up a read data sequence is:

- Request_To_Receive (RTR) (See 8.5)

The Final Destination controls the data flow with Control Operations. These Control Operations shall contain parameters specifying the Block number to send (B_num), and the buffer index (Bufx) and Offset parameters specifying where the data will be placed in the Final Destination. An RTS_Response/CTS Operation combines the RTS_Response and CTS Operations into one Operation.

- Clear_To_Send (CTS) (See 8.3.)
- RTS_Response/CTS (See 8.4.)

The STUs carrying the data payload shall consist of 40-byte Schedule Headers followed by data payload. Flags shall indicate if this is the first STU of a Block, if Concatenation is to be used, if the ULP should be notified, and if state information should be sent.

- DATA (See 8.6.)

Control Operations are used to acquire state information. The state information includes whether the specified Block was received correctly, the highest numbered Block of this Transfer that has been received correctly, and the number of available Slots in the Final Destination's Schedule Header queue. State information can be requested in a DATA Operation, or with a State_Request Control Operation.

- State_Request (See 8.7.)
- State_Response (See 8.8.)

The Control Operations below are used to abort a Transfer. Unlimited size Transfers shall use this method to signal the end of the Transfer.

- End (See 8.9.)
- End_ACK (See 8.10.)

4.4.2 Transfers

Transfers shall be assigned transfer identifiers (i.e., S_id and D_id) unique to each end device. For the source of an Operation, the transfer identifier (S_id) is its own local transfer identifier, and the D_id is the other end's local transfer identifier.

When Concatenate = 1, the Bufx and Offset fields shall be concatenated into a single 64-bit address (see 6.2).

| |
|---|
| <i>Open Issue – This text needs to be expanded to include Source_Concatenate if that bit is kept.</i> |
|---|

4.4.3 Blocks

Scheduled Transfer flow control, striping, acknowledgments, and resource allocation are all done on a Block basis. All Blocks of a Scheduled Transfer shall be the same size, except for the first and/or last Block of the Transfer, which may be smaller. Block numbers (B_num) shall be numbered starting at zero, and incrementing by one for each following Block.

Blocks which have been enabled for transfer may be delivered in any order. Receivers shall request that the Block be transmitted in sequential order, (i.e., using CTS Operations), unless pre-arrangements have been made to assure that the senders are capable of out-of-order transmission (these pre-arrangements are beyond the scope of this standard).

4.4.4 STUs

The STUs of a Block shall be transmitted in order. STU numbers (S_count) shall start with (number of STUs in this Block – 1) and decrement by one for each following STU. The last STU of a Block shall have S_count = 0. No STU shall extend past a Final Destination's buffer region boundary, Block size boundary, or Transfer boundary.

4.4.5 Bufx and Offset

When Concatenate = 0, Bufx specifies a buffer index for placing the data in the Final Destination. If more than one Bufx is required for a Block, i.e., Block size > buffer size, then the Bufx parameter in the CTS Operation shall specify the initial Bufx and any additional Bufx values shall be contiguous. Offsets may be used to start at other than the first byte of a buffer.

5 Service interface

This clause specifies the services provided by HIPPI-ST. The intent is to allow ULPs to operate correctly with this HIPPI-ST. How many of the services described herein are chosen for a given implementation is up to that implementor; however, a set of HIPPI-ST services must be supplied sufficient to satisfy the ULP(s) being used. The services as defined herein do not imply any particular implementation, or any interface.

Figure 4 shows the relationship of the HIPPI-ST interfaces.

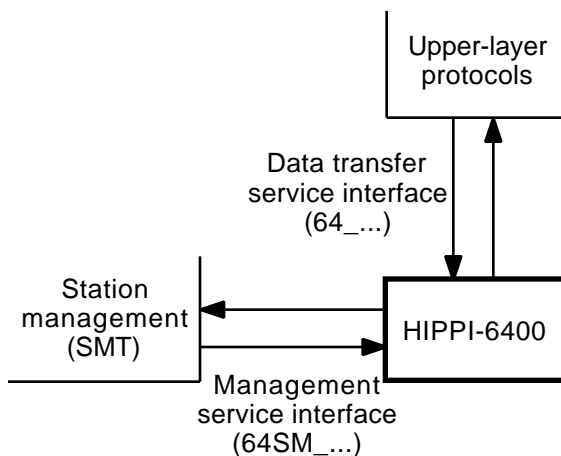


Figure 4 – HIPPI-ST service interface

5.1 Service primitives

The primitives, in the context of the state transitions in clause 5, are declared required or optional. Additionally, parameters are either required, conditional, or optional. All of the primitives and parameters are considered as required except where explicitly stated otherwise.

HIPPI-ST service primitives are of four types.

- *Request primitives* are issued by a service user to initiate a service provided by the HIPPI-ST. In this standard, a second Request primitive of the same name shall not be issued until the Confirm for the first request is received.

- *Confirm primitives* are issued by the HIPPI-ST to acknowledge a Request.

- *Indicate primitives* are issued by the HIPPI-ST to notify the service user of a local event. This primitive is similar in nature to an unsolicited interrupt. Note that the local event may have been caused by a service Request. In this standard, a second Indicate primitive of the same name shall not be issued until the Response for the first Indicate is received.

- *Response primitives* are issued by a service user to acknowledge an Indicate.

5.2 Sequences of primitives

The order of execution of service primitives is not arbitrary. Logical and time sequence relationships exist for all described service primitives. Time sequence diagrams are used to illustrate a valid sequence. Other valid sequences may exist. The sequence of events between peer users across the user/provider interface is illustrated. In the time sequence diagrams the HIPPI-ST users are depicted on either side of the vertical bars while the HIPPI-ST acts as the service provider.

NOTE - The intent is to flesh out the service primitives similar to what is in HIPPI-PH today.

6 Schedule Header

The Schedule Header shall be as shown in figure 5 as a group of 32-bit words. The Schedule Header shall be used with all Scheduled Transfers. The primary usage for each parameter is listed below, other uses are summarized in tables 1 and 2.

| | | | Bytes |
|------------------------------|-------|---------|-------|
| Op | Flags | S_count | 00-03 |
| D_Port | | S_Port | 04-07 |
| Key | | | 08-11 |
| D_id | | S_id | 12-15 |
| Bufx | | | 16-19 |
| Offset | | | 20-23 |
| T_len | | | 24-27 |
| B_num | | | 28-31 |
| 8 bytes of mandatory payload | | | 32-39 |

Figure 5 – Schedule Header contents

6.1 Schedule Header parameters

The Schedule Header parameters shall be as follows. If an Operation does not use a particular Schedule Header parameter, then that parameter shall be transmitted as zeros.

Op (5 bits, high-order 5 bits of byte 00) – The Scheduled Transfer Operation. See tables 1 and 2 for a summary of Op values. Unspecified values of Op are reserved.

Flags (11 bits, low-order 3 bits of byte 00, and all of byte 01) = Control flags (see 7.5).

S_count (16 bits, bytes 02-03) – The number of maximum-size STUs in a Block, or used as an STU counter (counting down to zero) in DATA Operations.

D_Port (16 bits, bytes 04-05) – The Final Destination's logical port for this Operation.

S_Port (16 bits, bytes 06-07) – The Original Source's logical port for this Operation.

Key (32 bits, bytes 08-11) – Virtual Connection identifier. Generated independently by each end during the Virtual

Connection setup. (See 8.2.)

D_id (16 bits, bytes 12-13) – The Final Destination's identifier for this Transfer. (See 8.1.)

S_id (16 bits, bytes 14-15) – The Originating Source's identifier for this Transfer. (See 8.1.)

Bufx (32 bits, bytes 16-19) – The buffer index at the Final Destination. Bufox is used at the Final Destination to select a memory location, e.g., from a table of buffer addresses, or as the high-order portion of a 64-bit concatenated address. (See annex C.)

Offset (32 bits, bytes 20-23) – The Final Destination's Offset within a Bufox, or the low-order portion of a 64-bit concatenated address. (See annex C.)

T_len (32 bits, bytes 24-27) – The length, in bytes, of the Transfer data. T_len = x'00000000' has special meaning, see 8.1.

B_num (32 bits, bytes 28-31) – Labels the Block being requested or acknowledged. B_num starts at zero. Note that Blocks may be transferred in any order (see annex C).

The eight bytes of mandatory payload (64 bits, bytes 32-39) are available for use by upper-layer protocols; their meaning is outside the scope of this standard.

6.2 Scheduled Transfer flags

Unspecified flag values are reserved. The flags shall be:

Open Issue – The flag bits were reordered in what seemed to be a more logical order, grouping related bits.

Interrupt (b'x1xxxxxxx') = Requests that the Final Destination immediately deliver this STU's Schedule Header to the appropriate upper-layer protocol upon successful receipt of this STU.

Notify (b'x1xxxxxxx') = Requests that the Final Destination deliver this STU's Schedule Header to the appropriate upper-layer protocol upon successful receipt of this STU.

State_Requested (b'xxx1xxxxxxx') = Requests that the Final Destination respond with a

State_Response upon successful receipt of this STU by the higher-layer protocol. For State_Requested to be valid, either the Interrupt or Notify flag must also = 1.

Source_Concatenate (b'xxxx1xxxxx') = The S_count, D_id, and B_num fields shall be concatenated into a single 64-bit Originating Source address. S_count shall contain the most-significant bytes of the address, and B_num the least-significant bytes. Support for its use is optional, and is indicated during Virtual Connection setup. Source_Concatenate is only used with RTR, (see 8.5).

Open Issue – Greg Chesson has an action item to determine if we need the Source_Concatenate bit or not, and if so the text to go with it.

Concatenate (b'xxxxx1xxxx') = The Bufx and Offset fields shall be concatenated into a single 64-bit Final Destination address. Bufx shall contain the most-significant bytes of the address. The value of the Concatenate flag shall be consistent for an entire Transfer, i.e., you cannot switch back and forth between 64-bit addresses and buffer indexes within a Transfer. Support for its use is optional, and is indicated during Virtual Connection setup.

Persistent (b'xxxxxx1xxxx'). When Persistent = 1, the memory in the Final Destination allocated for this Scheduled Transfer shall be retained for multiple transfers, and not released until Port_Tear down occurs. When Persistent = 0, the memory may be allocated for other uses after the Scheduled Transfer is complete. Support for its use is optional, and is indicated during Virtual Connection setup.

First (b'xxxxxxx1xxx') = First STU of a Block

Reject (b'xxxxxxxx1xx') = The request (e.g., RQP, RTS, or RTR) has been rejected.

Data channel assignment: (RTS only, see 8.1.)

b'xxxxxxxx00' = Receive the data on Data Channel 0.

b'xxxxxxxx01' = Receive the data on Data Channel 1.

b'xxxxxxxx10' = Receive the data on Data Channel 2.

b'xxxxxxxx11' = Receive the data on Data Channel 3.

7 Virtual Connection Operations

Ports are logical entities in the end devices. Virtual Connections are set up between two ports, called the A-Port and B-Port. The device that initiates the Virtual Connection is called device A, and the device at the other end is called device B. The Virtual Connection is full-duplex and symmetrical.

In addition to the port names, each port shall associate a Key value (A-Key and B-Key) with the Virtual Connection. Device A shall select the value of A-Key, and device B shall select the value of B-Key. Device B shall include the A-Key in all Operations. Device A shall use A-Key to validate the Operations.

Buffer sizes (A-Bufsize and B-Bufsize) shall be exchanged while setting up the Virtual Connection. The Bufsize parameter tells the other end the size, in bytes, of the device's buffers. If the sizes are not the same, then the data transfer shall use the smaller of the values as the maximum STU size on this Virtual Connection.

The maximum number (A-limit and B-limit) of unacknowledged Operations, (i.e., Schedule Headers) on this Virtual Connection that each device can support shall also be exchanged while setting up the Virtual Connection. The value specified shall be one less than the actual number of Slots so that a Slot for an End is always available. A-limit = x'FFFFFFF' means don't care, or that device A can support an unlimited number of Operations.

The Operations used to set up and tear down Virtual Connections are detailed below, and summarized in table 1. Only the fields used in each Operation are listed, all of the other Schedule Header fields shall be transmitted as zeros. While a particular field usually carries the parameter of the same name, fields sometimes carry other parameter values. In the Operations below, the specific parameter used in the Operation is listed first, and if is not carried in the field of the same name then the field name is included in square brackets.

7.1 Request_Port (RQP)

Request_Port shall be used to setup a Virtual Connection from this end device (called the A-Port) to another end device (called the B-Port). Device A shall also use the RQP Operation to inform device B about device A's buffer size, and assigns A-Key, A-Port, and A-limit values, and indicates support or not for Source_Concatenate, Concatenate, and Persistent.

Semantics – RQP (

- Op,
- Flags,
- A-limit [S_count],
- B-Port [D_Port],
- A-Port [S_Port],
- A-Bufsize [Bufx],
- A-Key [Offset],
- EtherType [B_num])

Op = x'01'

Flags shall specify the Source_Concatenate, Concatenate, and Persistent flags. A value of 1 shall indicate that device A supports that feature.

A-limit, carried in the S_count field, shall specify the maximum number of unacknowledged Operations, (i.e., Schedule Headers) that device A can support on this Virtual Connection. Operations received in excess of this number may be discarded. The value specified shall be one less than the actual number of Slots so that a Slot for an End is always available. A-limit = x'FFFFFFF' means don't care, or that device A can support an unlimited number of Operations.

B-Port, carried in the D_Port field, shall specify the logical port value for device B for this Virtual Connection. B-Port may be either a well-known port that is providing this service, or a peer port that provides this service. The default well-known port, when device B is expected to assign a port value, shall be D_Port = x'0000'. Device B shall decode the EtherType field to determine the Virtual Connection's data type.

A-Port, carried in the S_Port field, shall specify the logical port value for device A for this Virtual Connection. Device B shall use this A-Port value when replying to device A

concerning this Virtual Connection.

A-Bufsize, carried in the Bufx field, shall specify device A's buffer size.

A-Key, carried in the Offset field, shall be the Key value assigned by device A for this Virtual Connection. Device B shall include the A-Key in all Operations. Device A shall use A-Key to validate the Operations.

EtherType (see 7.2), carried in the B_num field, shall be a value that characterizes the data payloads that will be exchanged on this Virtual Connection.

Issued – By device A.

Effect – If it accepts the request, then device B shall establish a Virtual Connection and shall reply with an RQP_Response. Note that the Virtual Connection is full-duplex in that either device A or device B device may initiate a Scheduled Transfer. Multiple Scheduled Transfers may occur over a single Virtual Connection, and the Scheduled Transfers can be either writes or reads.

7.2 RQP_Response

RQP_Response shall inform device A whether the Virtual Connection was accepted or not. If accepted, device B shall use the RQP_Response Operation to inform device A about device B's buffer size, to assign B-Key, B-Port, and B-limit values, and to indicate support or not for Source_Concatenate, Concatenate, and Persistent.

Semantics – RQP_Response (

- Op,
- Flags,
- B-limit [S_count],
- A-Port [D_Port],
- B-Port [S_Port],
- A-Key [Key],
- B-Bufsize [Bufx],
- B-Key [Offset])

Op = x'02'

Flags shall specify the Source_Concatenate, Concatenate, Persistent, and Reject flags. Source_Concatenate, Concatenate, or Persistent =1 shall indicate that device B

supports that feature. If Reject = 1, then the Operation is refused. The actions taken when an RQP is refused are beyond the scope of this standard.

B-limit, carried in the S_count field, shall specify the maximum number of unacknowledged Operations, (i.e., Schedule Headers) that device B can support on this Virtual Connection. Operations received in excess of this number may be discarded. The value specified shall be one less than the actual number of Slots so that a Slot for an End is always available. B-limit = x'FFFFFFF' means don't care, or that device B can support an unlimited number of Operations.

A-Port, carried in the D_Port field, shall be the same as the A-Port value in the RQP Operation.

B-Port, carried in the S_Port field, shall contain the B-Port value assigned by device B for this Virtual Connection.

A-Key, carried in the Key field, shall be the validation key previously assigned by device A in the RQP Operation.

B-BuFSIZE, carried in the Bufx field, shall specify device B's buffer size. The smaller of A-BuFSIZE and B-BuFSIZE shall be used as the maximum STU size for this Virtual Connection.

B-Key, carried in the Offset field, shall be the Key value assigned by device B for this Virtual Connection. Device A includes the B-Key in all Operations. Device B uses B-Key to validate the Operations.

Issued – By device B in response to an RQP.

Effect – Unless disallowed by Reject = 1, device A has been assigned a logical port on device B. If accepted, the keys, buffer sizes, and maximum number of outstanding Control Operations have been exchanged, and a Virtual Connection has been established.

7.3 Port_Teardown

Port_Teardown shall terminate the Virtual Connection, and may be issued by either device A or device B. The Port_Teardown sequence uses a three-way handshake.

Open Issue – A more complete description of the 3-way handshake needs to be provided.

Semantics – Port_Teardown (

Op,
D_Port,
S_Port,
Key)

Op = x'03'

D_Port shall contain the value associated with the recipient of the Operation, e.g., D_Port = B-Port when the Port_Teardown is issued by device A.

S_Port shall contain the value associated with the initiator of the Operation, e.g., S_Port = A-Port when the Port_Teardown is issued by device A.

Key shall contain the Key value associated with the recipient of the Operation, e.g., Key = B-Key when the Port_Teardown is issued by device A.

Issued – By either side, i.e., device A or device B, of the Virtual Connection. The sender should not issue a Port_Teardown when there is unacknowledged data on the Virtual Connection.

Effect – The sender should release any buffers associated with this Virtual Connection, but shall retain the Port and Key values for use in further Port_Teardown Operations.

7.4 Port_Teardown_ACK

Port_Teardown_ACK shall be used to acknowledge receipt of a Port_Teardown.

Semantics – Port_Teardown_ACK (

Op,
D_Port,
S_Port,
Key)

Op = x'04'

D_Port shall contain the value associated with the recipient of the Operation, e.g., D_Port = B-Port when the Port_Teardown_ACK is issued by device A.

S_Port shall contain the value associated with the initiator of the Operation, e.g., S_Port = A-Port when the Port_Teardown_ACK is issued by device A.

Key shall contain the Key value associated with the recipient of the Operation, e.g., Key = B-Key when the Port_Teardown_ACK is issued by device A.

Issued – By the recipient of a Port_Teardown Operation after releasing this Virtual Connection's buffers.

Effect – TBD

7.5 Port_Teardown_Complete

Port_Teardown_Complete shall be used to provide a three-way handshake, acknowledging that the actions associated with a Port_Teardown have been completed.

Semantics – Port_Teardown_Complete (

Op,
D_Port,
S_Port,
Key)

Op = x'05'

D_Port shall contain the value associated with the recipient of the Operation, e.g., D_Port = B-Port when the Port_Teardown_Complete is issued by device A.

S_Port shall contain the value associated with the initiator of the Operation, e.g., S_Port = A-Port when the Port_Teardown_Complete is issued by device A.

Key shall contain the Key value associated with the recipient of the Operation, e.g., Key = B-Key when the Port_Teardown_Complete is issued by device A.

Issued – By the recipient of a Port_Teardown_ACK Operation.

Effect – After the Op-retry expires, the sender shall release this Virtual Connection's Port and Key values. The recipient of the Port_Teardown_Complete shall release this Virtual Connection's Port and Key values.

8 Scheduled Transfer data Operations

The Operations used for Scheduled Transfers are detailed below, and summarized in table 2. All of the Scheduled Transfer data transfer Operations use the D_Port, S_Port, A-Key, and B-Key values that were assigned during the Virtual Connection setup (see 8.2). When device A issues the Operation then S_Port = A-port, D_Port = B-port, and Key = B-Key. Likewise, when device B issues the Operation then S_Port = B-port, D_Port = A-port, and Key = A-Key. For clarity and brevity, these values are not listed in the individual Operations. All other Schedule Header parameters that are not listed in a specific Operation shall be transmitted as zeros.

8.1 Request_To_Send (RTS)

Request_To_Send is issued by the Originating Source to specify the number of data bytes (T_len) to be sent from the Originating Source (S_Port) to the Final Destination (D_Port), and the Source's Transfer identifier (T-id). In addition, the Originating Source shall specify whether 64-bit address or buffer indexes are used, whether the Final Destination's buffer should be maintained or discarded after the transfer, and the Data Channel assignment for the data transfer. Note that the device on either end of the Virtual Connection may issue a Request_To_Send.

Semantics – RTS (

Op,
Flags,
S_id,
T_len)

Op = x'06'

Flags shall specify the Concatenate, Persistent, and Data Channel assignment flags (see 7.5). Concatenate or Persistent shall only = 1 if the corresponding flag was set = 1 by the other end during the Virtual Connection setup (see 7.1 and 7.2).

S_id shall be the Transfer identifier used by this end (i.e., the Originating Source) to identify this Transfer. The Final Destination

shall use this value as the D_id parameter when replying to the Originating Source concerning this Transfer.

T_len shall specify the total number of data payload bytes in the Transfer. T_len does not include the Schedule Header, or any lower-layer headers. T_len = x'00000000' shall mean an unlimited size Transfer. An unlimited size Transfer is terminated by an End Operation (see 8.7).

Issued – By the Originating Source after a Virtual Connection has been established.

Effect – The Final Destination shall setup for the data transfer, and then reply back to the Originating Source with the associated parameters.

8.2 RTS_Response

RTS_Response shall be used to tell the Originating Source if the transfer is allowed or not. If allowed, the RTS_Response specifies the Transfer identifier assigned by this end (i.e., the Final Destination) for this Transfer, and the number of STUs per Block. An RTS_Response does not give the Originating Source permission to start transmission; that comes from a CTS.

Semantics – RTS_Response (

- Op,
- Flags,
- Blocksize [S_count],
- D_id,
- S_id,
- T_len)

Op = x'07'

Flags shall specify the Reject and Concatenate flags. If Reject = 1, then the Operation is refused. The actions taken when an RTS is refused are beyond the scope of this standard. Concatenate shall be the same value as in the RTS.

Blocksize, carried in the S_count field, shall specify the number of maximum size STUs that may be transmitted in each Block.

D_id shall be the Transfer identifier assigned by the Originating Source in the RTS

Operation.

S_id shall be the Transfer identifier used by the Final Destination to identify this Transfer. The Originating Source shall use this value as the D_id parameter when replying to the Final Destination concerning this Transfer.

T_len shall be the same value as in the RTS Operation, but it need not be checked by the Originating Source.

Issued – By the Final Destination.

Effect – The Originating Source shall segment the Transfer into Blocks and STUs. (See figure 2 and annex C.)

8.3 Clear_To_Send (CTS)

Clear_To_Send shall be used to give the Originating Source permission to send one Block. Clear_To_Send may also be used to request retransmission of a Block from systems that are capable of retransmissions.

Semantics – CTS (

- Op,
- Flags,
- D_id,
- S_id,
- Bufox,
- Offset,
- B_num)

Op = x'08'

Flags shall specify the Concatenate flag. Concatenate shall be the same value as in the RTS or RTR that initiated this Transfer.

D_id shall be the Transfer identifier assigned by the other end of the Virtual Connection.

S_id shall be the Transfer identifier assigned by this end of the Virtual Connection.

Bufox shall specify the initial buffer index in the Final Destination where the data will be placed. If multiple Bufox values are required for a Block, then the Bufox values shall be contiguous.

Offset is a value that the Final Destination wants to receive with the first STU so that the data can be properly placed in the Final Destination's memory.

B_num shall be the Block number being given permission to be transmitted. Block numbers shall start at zero and increment by one for subsequent Blocks.

Issued – By the Final Destination.

Effect – The Originating Source shall transfer the specified Block.

8.4 RTS_Response/CTS

RTS_Response/CTS is a combined RTS_Response and a Clear_To_Send, i.e., it tells the Originating Source how to break up the Transfer, and also gives permission to send one Block.

Semantics – RTS_Response/CTS (

- Op,
- Flags,
- Blocksize [S_count],
- D_id,
- S_id,
- Bufox,
- Offset,
- T_len,
- B_num)

Op = x'09'

Flags shall specify the Concatenate flag. Concatenate shall be the same value as in the RTS that initiated this Transfer.

Blocksize, carried in the S_count field, shall specify the number of maximum size STUs that may be transmitted in each Block.

D_id shall be the Transfer identifier assigned by the other end of the Virtual Connection.

S_id shall be the Transfer identifier assigned by this end of the Virtual Connection.

Bufox shall specify the initial buffer index in the Final Destination where the data will be placed. If multiple Bufox values are required for a Block, then the Bufox values shall be contiguous.

Offset is a value that the Final Destination wants to receive with the first STU so that the data can be properly placed in the Final Destination's memory.

T_len shall be the same value as in the RTS

Operation, but it need not be checked by the receiver.

B_num shall be the Block number being given permission to be transmitted. Block numbers start at zero and increment by one for subsequent Blocks.

Issued – By the Final Destination.

Effect – The Originating Source shall segment the Transfer into Blocks and STUs, and shall transfer the specified Block.

8.5 Request_To_Receive (RTR)

Request_To_Receive is issued by the Final Destination to specify the number of data bytes (T_len) to be sent from the Originating Source to the Final Destination, and the Final Destination's Transfer identifier. In addition, the Final Destination shall specify whether 64-bit addresses or buffer indexes are used at the Originating Source, at the Final Destination, or at both. An RTR transfers a single Block; there is no notion of a multi-Block RTR transfer. When an RTR is issued it is assumed that the ULPs on both end devices had previously allocated resources for the entire Transfer, e.g., through a previous RTS Operation. Note that the device at either end of the Virtual Connection may issue a Request_To_Receive.

Semantics – RTR (

- Op,
- Flags,
- OS-Offset [S_count | D_id],
- S_id,
- Bufox,
- Offset,
- T_len,
- OS_Bufox [B_num])

Op = x'0A'

Flags shall specify the Concatenate flag (for controlling addressing on the Final Destination), and the Source_Concatenate flag (for controlling the addressing on the Originating Source). Source_Concatenate shall only = 1 if Source_Concatenate = 1 was set by this device during the Virtual Connection setup (see 7.1 and 7.2).

OS_Offset, carried in the concatenation of

the S_count and D_id fields (yes, these are two disjoint 16-bit fields), shall specify the Originating Source's offset value. The S_count field shall contain the most-significant bytes of OS_Offset.

S_id shall be the Transfer identifier used by the Final Destination to identify this Transfer. The Originating Source shall use this value as the D_id parameter when replying to the Final Destination concerning this Transfer.

Bufox shall specify the Final Destination's buffer index.

Offset shall be the Offset where the Final Destination wants to receive with the first STU so that the data can be properly placed in the Final Destination's memory.

T_len shall specify the total number of data payload bytes in the Transfer. T_len does not include the Schedule Header, or any lower-layer headers.

OS_Bufox, carried in the B_num field, shall specify the Originating Source's buffer index.

Issued – By the Final Destination.

Effect – The Originating Source shall transfer the specified Transfer.

8.6 DATA

A DATA Operation transfers an STU of a Block from the Originating Source to the Final Destination. No STU shall be larger than the maximum STU size determined during the Virtual Connection setup (see 7.2), plus 32 bytes of Schedule Header.

Semantics – DATA (

Op,
Flags,
S_count,
D_id,
S_id,
Bufox,
Offset,
Sync [T_len],
B_num)

Op = x'0B'

Flags shall specify the Interrupt, Notify, State_Requested, Concatenate, and First flags (see 7.5). Concatenate shall be the same value as in the RTS or RTR that initiated this Transfer. State_Requested may be sent with any STU of a Block. The State_Response Control Operation associated with this request shall be issued after processing this STU.

S_count shall be the STU number. S_count shall be decremented before each STU is sent. The first STU shall have S_count = (number of STUs in the Block) – 1. The last STU in a Block shall have S_count = x'0000'.

D_id shall be the Transfer identifier assigned by the other end of the Virtual Connection.

S_id shall be the Transfer identifier assigned by this end of the Virtual Connection. Note that if this is the first STU associated with an RTR Operation, then this Transfer identifier is being assigned by the Originating Source, and shall be used by the Final Destination as the D_id parameter when replying to the Originating Source concerning this Transfer.

Bufox shall be the buffer index at the Final Destination. The first STU of the Block shall use the Bufox value received in the CTS, RTS_Response/CTS, or RTR, Operation. Subsequent STUs of the Block shall adjust the Bufox value based on the Final Destination's buffer size. (See annex C.)

Offset shall be the Final Destination's offset within a Bufox. Offset is satisfied within the first STU of a Block, and hence zero for subsequent STUs of the Block, except when the Originating Source has a smaller buffer size than the Final Destination. (See annex C.)

Sync, carried in the T_len field, shall be a value assigned by the Originating Source to synchronize the current view of the number of Operations that can be accepted by the Final Destination. The maximum number of unacknowledged Operations was established by the A-limit and B-limit exchange during the Virtual Connection setup (see 7.1 and 7.2). The current value of empty Slots is returned to the Originating Source by a State_Response Control Operation (see 8.8). The Sync value is only valid when State_Requested = 1.

B_num shall be the number of the Block that this STU is a part of.

Issued – By the Originating Source.

Effect – The Final Destination shall place the STU data in the memory area pointed to by Bufx, and offset by the Offset value. The Final Destination shall only accept data into pre-allocated buffer regions. The Final Destination is responsible for ensuring that all of the Blocks of a Transfer are received. The actions to be taken if a Block is missing are beyond the scope of this standard.

8.7 State_Request

State_Request is used to request that the other end provide its current number of empty Slots for Schedule Headers, the Block number associated with the last set of contiguously good data received, and whether the named Block was received correctly.

Semantics – State_Request (

Op,
D_id,
S_id,
Sync [T_len],
B_num)

Op = x'0C'

D_id shall be the Transfer identifier assigned by the other end of the Virtual Connection. D_id = x'FFFF' means that the receiver shall not look for a current Transfer, and only return the current number of empty Slots for this Virtual Connection.

S_id shall be the Transfer identifier assigned by this end of the Virtual Connection. If D_id = x'FFFF', then S_id shall also be x'FFFF'.

Sync, carried in the T_len field, shall be the same as defined in the DATA Operation (see 8.6).

Issued – By an end device that needs state information from the device on the other end of the Virtual Connection. The sender may not have received the State_Responses it expected, and can use State_Requests to recover from lost or damaged State_Responses.

Effect – The receiver shall immediately reply with a State_Response. Note that the State_Response associated with the State_Requested flag bit in an DATA Operation waits for the STU to finish processing rather than immediately.

8.8 State_Response

State_Response shall be used to indicate the number of empty Slots in this Port of the Virtual Connection. State_Response may also indicate the highest numbered contiguous Block received correctly, and whether the named Block was received correctly.

Semantics – State_Response (

Op,
C-limit [S_count],
D_id,
S_id,
B_seq [Offset],
Sync [T_len],
B_num)

Op = x'0D'

C-limit, carried in the S_count field, shall indicate the sender's view of the number of empty Slots it has available for additional Operations. (See 8.6.)

D_id shall echo the S_id value in the State_Request or DATA Operation that triggered this State_Response.

S_id shall echo the D_id value in the State_Request or DATA Operation that triggered this State_Response. S_id = x'FFFF' shall mean that the B_seq and B_num parameters are meaningless.

B_seq, carried in the Offset field, shall indicate the highest numbered contiguous Block received correctly.

Sync, carried in the T_len field, shall be the Operation identifier in the sender's Operation stream where the C-limit value was determined. Since the receiver can count the number of Operations sent since this Sync value, it now knows how many more Operations it can send without overflowing, (i.e., overflow may result in the extra Operations being discarded). Note that C-

limit is the minimum value, additional empty Slots may accumulate during the processing and propagation time.

B_num shall be the number of the Block being acknowledged. A value of x'FFFFFFFF' shall indicate that no complete Blocks have been received.

Issued – It is intended that a State_Response be issued by an end device's higher-layer protocol after receiving State_Requested = 1 in a DATA Operation, or receiving a State_Request Operation.

Effect – State information is passed to the other end of the Virtual Connection.

8.9 End

End allows either end of the Virtual Connection to terminate a Scheduled Transfer before it has completed, and to terminate Scheduled Transfers of unlimited size.

Semantics – End (

Op,
D_id
S_id)

Op = x'0E'

D_id shall be the Transfer identifier assigned by the other end of the Virtual Connection.

S_id shall be the Transfer identifier assigned by this end of the Virtual Connection. This S_id value shall not be reused until an End_ACK is received.

Issued – By the Originating Source or the Final Destination.

Effect – A Final Destination receiving an End shall stop sending Control Operations associated with this Scheduled Transfer. An Originating Source receiving an End shall stop sending Control Operations and STUs associated with this Scheduled Transfer. An End kills a Scheduled Transfer, but shall not affect the Virtual Connection carrying the Scheduled Transfer.

8.10 End_ACK

End_ACK confirms that this end device has seen, and acted on, the End.

Semantics – End_ACK (

Op,
D_id,
S_id)

Op = x'0F'

D_id shall be the Transfer identifier assigned by the other end of the Virtual Connection.

S_id shall be the Transfer identifier assigned by this end of the Virtual Connection. This S_id value should not be immediately reused to avoid aliasing.

Issued – By the end of the Virtual Connection that received the End Operation.

Effect – Acknowledgment that the Scheduled Transfer has been terminated.

9 Sender detected errors

For Operations that have an expected response, if the response is not received within an Op-retry period, then the original Operation shall be retransmitted to recover from a lost or damaged Operation. Expected responses to Operations are:

- RQP_Response to RQP
- Port_Teardown_ACK to Port_Teardown
- Port_Teardown_Complete to Port_Teardown_ACK
- RTS_Response to RTS
- RTS_Response/CTS to RTS
- End_ACK to End

The Op-retry period is system dependent and shall be determined by:

- a time longer than the measured round-trip time through the software path; or
- a long fixed time period; or
- consumption of all of the T_id values; or
- the number of unacknowledged Operations consumed too many Final Destination resources.

Table 1 – Virtual Connection Operations summary

| Operation | Op | Flags | S_count | D_Port | S_Port | Key | Bufx | Offset | B_num |
|--|-------|--------------|----------------|---------------|---------------|-------|------------------|--------------|------------------|
| Request_Port (RQP) (<i>from A</i>) | x'01' | OC P | A-limit | B-Port | A-Port | * | A-Bufsize | A-Key | EtherType |
| RQP_Response (<i>from B</i>) | x'02' | OC PR | B-limit | A-Port | B-Port | A-Key | B-Bufsize | B-Key | * |
| Port_Teardown | x'03' | * | * | D_Port | S_Port | D-Key | * | * | * |
| Port_Teardown_ACK | x'04' | * | * | D_Port | S_Port | D-Key | * | * | * |
| Port_Teardown_Complete | x'05' | * | * | D_Port | S_Port | D-Key | * | * | * |
| NOTES – 1 – Flag abbreviations are: O = Source_Concatenate, C = Concatenate, P = Persistent, R = Reject 2 – D-Key = Key value the Destination binds to, e.g., D-Key = A-Key when Operation issued by device B. 3 – D_Port = Port number in device receiving the Operation, e.g., D_Port = A-Port when issued by device B. 4 – S_Port = Port number in device issuing the Operation, e.g., S_Port = B-Port when issued by device B. 5 – The Schedule Header parameters that are not shown shall be transmitted as zeros. SYMBOLS - * = Unused value, transmit as 0 Values in bold italics are assigned by the specific Operation, and may be used by later Operations | | | | | | | | | |

Table 2 – Data transfer Operations summary

| Operation | Op | Flags | S_count | D_id | S_id | Bufx | Offset | T_len | B_num |
|---|-------|---------------|------------------|------|-------------|-------------|---------------|--------------|----------------|
| Req_To_Send (RTS) (<i>from G</i>) | x'06' | CP V | * | * | G_id | * | * | T_len | * |
| RTS_Response (<i>from H</i>) | x'07' | C R | Blocksize | G_id | H_id | * | * | T_len | * |
| Clear_To_Send (CTS) | x'08' | C | * | D_id | S_id | Bufx | Offset | * | B_num |
| RTS_Response/CTS | x'09' | C | Blocksize | D_id | S_id | Bufx | Offset | T_len | B_num |
| Req_To_Receive (RTR) (<i>from G</i>) | x'0A' | OC | OS-Offset | | G_id | Bufx | Offset | T_len | OS-Bufx |
| DATA | x'0B' | IN SCF | S_count | D_id | S_id | Bufx | Offset | Sync | B_num |
| State_Request | x'0C' | * | * | D_id | S_id | * | * | Sync | B_num |
| State_Response | x'0D' | * | C-limit | D_id | S_id | * | B_seq | Sync | B_num |
| End | x'0E' | * | * | D_id | S_id | * | * | * | * |
| End_ACK | x'0F' | * | * | D_id | S_id | * | * | * | * |
| NOTES – 1 – Flag abbreviations are: I = Interrupt, N = Notify ULP, S = State_Requested, O = Source_Concatenate, C = Concatenate, P = Persistent, F = First STU of Block, R = Reject, V = Data channel assignment 2 – D_id = Transfer identifier in device receiving the Operation, e.g., D_id = G_id when issued by device H. 3 – S_id = Transfer identifier in device issuing the Operation, e.g., S_id = H_id when issued by device H. 4 – Schedule Header parameters that shall be transmitted as assigned in RQP and RQP_Response Operations: D_Port = Port number of the device receiving the Operation S_Port = Port number of the device issuing the Operation Key = Key value assigned by the device receiving the Operation SYMBOLS - * = Unused value, transmit as 0 Values in bold italics are assigned by the specific Operation, and may be used by later Operations | | | | | | | | | |

10. Receiver detected errors

Open Issue – All errors and processing are new - please review. Please suggest any missing errors.

10.1 General errors

10.1.1 Undefined Opcode

An undefined Opcode value may occur due to bit errors, or if the sending device is using a future superset of the Scheduled Transfer Operations. The Operation shall be discarded and an Undefined_Opcode_Error shall be logged.

Open Issue – Should the offending Opcode be logged?

10.1.2 Unexpected Opcode

Most of the Operations require previous Operations to setup state on each device. If a device receives an out of sequence Opcode, (e.g., receiving an RQP_Response without issuing the initiating RQP), the Operation shall be discarded and an Unexpected_Opcode_Error shall be logged.

10.2 Virtual Connection errors

10.2.1 Invalid Key or Port

All Operations, excluding RQP, should have a Key value that validates the Operation for the Virtual Connection. Operations with an invalid key shall not be executed, and an Invalid_Key_Error shall be logged.

All Operations should have a valid Destination port value. All Operations, excluding RQP, should have a Source port value that matches the state for this Virtual Connection. Operations with an invalid port shall not be executed, and an Invalid_Port_Error shall be logged.

NOTE – Multiple contiguous invalid Key and/or port values may indicate a problem with the link, or a malicious host on the network. The supervising process should be informed.

10.2.2 Op-limit exceeded

Operations that exceed the Op-limit for the Virtual Connection shall not be executed, and an Op-limit_Exceeded_Error shall be logged.

10.2.3 Unknown EtherType

If an RQP Operation contains an unknown EtherType, the receiver shall issue an RQP_Response with the Reject bit set and log an Unknown_EtherType_Error.

Open Issue – Should the Operation above be rejected or discarded?

10.2.4 Illegal BuFSIZE

If an RQP or RQP_Response contains a BuFSIZE value that is less than 32 bytes, or is not a power of 2, then the receiver shall not execute the Operation, and shall log an Illegal_BuFSIZE_Error.

10.3 Scheduled Transfer errors

10.3.1 Invalid id

All Scheduled Transfer Operations, except RTS and RTR, should have a valid Destination id for quickly accessing state information for this Scheduled Transfer. After checking the D_id, the S_id should match the stored value for this Transfer. An invalid id shall result in not executing the Operation, and logging an Invalid_id_Error.

10.3.2 Bad Data Channel specification

During an RTS Operation, the sending device declares the lower layer Data Channel that will carry DATA Operations for this Scheduled Transfer. Some Data Channels may not be used for Scheduled Transfers depending on the lower layer, (e.g., b'00' is not a valid choice on HIPPI-6400 as it indicates VC0 which is reserved for Control Operations). The receiver shall issue an RTS_Response with the Reject bit set.

10.3.3 Concatenate not available

If the Virtual Connection did not specify the capability for Concatenation, any Scheduled Transfer Operations on this Virtual Connection with the Concatenate bit set shall not be executed.

10.3.4 Block size exceeds Transfer length

If a RTS_Response, or RTS_Response/CTS, is received specifying a Block size larger than the Transfer length, then the Originating Source shall treat the Transfer as a single Block.

10.3.5 Out of Range B_num, Bufx, Offset, S_count, or Sync

During the CTS, RTS_Response/CTS, DATA, and State_Response Operations a Block number may appear that is outside the calculated number of Blocks for the Transfer. If an out of range Block number is encountered, the receiver shall not execute the Operation, and shall log an Out_Of_Range_B_num_Error.

If a DATA Operation contains a Bufx and/or Offset that exceeds the buffer range allocated by the Final Destination for previous outstanding CTS's, then the receiver shall not execute the Operation, and shall log an Out_Of_Range_Bufx_Error.

If a DATA Operation contains an Offset larger than the buffer size, the receiver shall not execute the Operation, and shall log an Oversized_Offset_Error.

If a DATA Operation without the "First STU of Block" flag bit contains an S_count that is not one less than the previous STU (for this Scheduled Transfer), then the STU is out of order. The receiver shall discard the STU and log an Out_of_Order_STU_Error.

If a State_Response Operation is received with an out of range Sync parameter, the Operation shall not be executed, and an Out_Of_Range_Sync error shall be logged.

10.3.6 RTR problem

The RTR Operation should be setup by the higher layer. If an RTR Operation is received and a problem occurs, the receiver shall issue an RTS_Response with the appropriate fields (including the id from the RTR in the D_id) and set the Reject bit.

10.4 Other errors

Various timeout errors should be covered in the higher layer implementation, (e.g., the timeout associated with issuing a CTS Operation but lacking receipt of any corresponding DATA Operations.)

Table 3 – Summary of logged errors

| Name | Occurs in Operation |
|--------------------------|---|
| Undefined_Opcode_Error | not applicable |
| Unexpected_Opcode_Error | all except RQP |
| Invalid_Key_Error | all except RQP |
| Invalid_Port_Error | all |
| Op-limit_Exceeded_Error | all with Opcode ≥ 6 |
| Unknown_EtherType_Error | RQP |
| Illegal_Bufsize_Error | RQP, RQP_Response |
| Invalid_id_Error | all with a D_id |
| Out_Of_Range_B_num_Error | CTS, DATA, State_Response, RTS_Response/CTS |
| Out_Of_Range_Bufx_Error | DATA |
| Oversized_Offset_Error | DATA |
| Out_of_Order_STU_Error | DATA |
| Out_of_Range_Sync_Error | State_Response |

Annex A
(normative)

Using HIPPI-6400-PH as the lower layer

Open Issue – This is new, and just a collection of random thoughts for now. It will be fleshed out and polished as time permits.

The data units passed between HIPPI-ST and HIPPI-6400-PH are STUs.

Somewhere we will have to say something about the padding, e.g., any padding received by HIPPI-6400-PH will be passed to -ST. If -PH required padding in the last micropacket, then it is supplied by -ST. The length value passed between -PH and -ST will be the -PH payload length, i.e., the Schedule Header + data payload.

-ST will need to supply the ULAs, or some address that -PH can use to derive the ULAs, to -PH. EtherType will also have to be passed.

Currently the document is written where -ST can request which -PH VC to use for data. Do we want to continue this?

Annex B
(normative)

Using HIPPI-FP as the lower layer

Open Issue – This is new, and just a collection of random thoughts for now. It will be fleshed out and polished as time permits.

HIPPI Packets will probably map to -ST Transfers. Hence, we need to expose the -ST Transfer to the lower layer, where to HIPPI-6400-PH we just exposed STUs.

Do we need some way to force an error on the -ST side to map the case of receiving a HIPPI packet in error?

We can probably do striping based on the low order bits of Bufx.

Annex C

(informative)

Scheduled Transfer example

The following example demonstrates the Scheduled Transfer Operations. In this example device X is sending a 4.82 MB file to device Y.

- Boxed equations showing C coding calculations are shown at the end of each Operation.
- Fields that contain a * are unused and transmitted as zeroes.
- Please note for this section that kB stands for 1024 bytes and MB stands for 1024 x 1024 bytes.
- All hex values that take up fewer bits than their field have zeros in the upper bits.
- Table C.1 shows all fields and corresponding values for the Operations shown below. The Operation description and table location are correlated by the numbers in parenthesis.

C.1 Virtual Connection setup

Device Y initiates a Virtual Connection setup request (RQP) with device X. The Virtual Connection setup is a dual direction setup and either device can begin a Scheduled Transfer after setup independent of who started the Virtual Connection setup.

Y->X (1)

RQP(S_count: Op_limitY, D_Port: x'0000', S_Port: portY, Bufx: BufsizeY (16 kB), Offset: KeyY, B_num: EtherType)

Y provides its Key, port, Op_limit and Bufsize to device X. X binds to the well-known port x'0000' which is used for port setup. Device X responds to the request with an RQP_Response.

BufsizeY = 16; /* kB */

X->Y (2)

RQP_Response(S_count: Op_limitX D_Port: portY, S_Port: portX, Key: KeyY, Bufx: BufsizeX (4 kB), Offset: KeyX,)

Device X provides device X's Key, port, Op_limit, and Bufsize to device Y. Y binds to the KeyY and portY to associate the response with the above RQP (as opposed to other RQP's that Y may have initiated).

A dual direction Virtual Connection has been established and both sides know the other's Key, port, Op_limit and Bufsize. The Keys are an authentication value which will stop invalid Operations, (e.g., an inadvertent Port_Teardown Operation might destroy the Virtual Connection.) The port values are used to map to higher layer protocols and may be the same mapping used for Internet style ports. Op_limit gives the destination some flow control power over Virtual Connection Operations. The number of unacknowledged Operations sent to the other device may not exceed Op_limit. State_Response Control Operations will provide the a current value of Op_limit and can be synchronized using the Sync field. The maximum STU size is determined by the smaller of the two devices' Bufsize values (4kB in this example).

```
BufsizeX = 4; /* kB */
if(BufsizeX < BufsizeY)
    max_Msg_size = BufsizeX;
else max_Msg_size = BufsizeY;
/* max_Msg_size = 4kB */
```

C.2 Scheduled Transfer setup

Device X initiates a transfer of 4.82 MB using the Virtual Connection established above.

X->Y (3)

RTS(Flags: x'02', D_Port: portY, S_Port: portX, Key: KeyY, S_id: idX, T_len: x'4D1EB8')

Device Y expects the data to be delivered on Data Channel 2 as specified in the Flags parameter. Device Y checks the Originating Source ID (idX) with it's list of acceptable devices. Device Y reads the transfer length size (~4.82 MB) and agrees to begin the transfer, but because device Y needs time to pin down buffers, only an RTS_Response is sent.

```
T_len = 0x4D1EB8; /* 5,054,136 bytes */
```

Y->X (4)

RTS_Response(S_count: 5, D_Port: portX, S_Port: portY, Key: KeyX, D_id: idX, S_id: idY, T_len: x'4D1EB8')

Device Y has accepted the transfer by not setting the reject bit. Device Y chose a nominal Block size of 5 maximally sized STUs. Device Y has assigned an ID (idY) which it can use to quickly index to the correct transfer. All further Scheduled Transfer Operations will continue to use the appropriate binding information: Key, ports, and the ID's but they are not shown in the remaining steps of this example.

Both devices calculate the nominal Block length.

```
S_count = 5;
nomBlk_len = max_Msg_size * S_count;
/* nomBlk_len = 20 kB */
```

C.3 Block 0 transfer

Device Y sends a Clear-to-Send Operation once it has finished allocating resources for the transfer.

Y->X (5)

CTS(Bufx: x'320', Offset: x'400', B_num: x'0')

The CTS gives us the Destination's buffer index (Bufx) and the Offset from which to begin filling the Bufx. The Scheduled Transfer Operations concentrate required complexity on the Originating Source (device X in this case). All Block sizes must be the same except for the first (which will be short by the Offset value) and the last (which will finish off the transfer). In this

example, the first block will be 19kB (20kB Block size - 1kB Offset). The 1kB Offset will force an offset between the two devices and hence tiling will be mismatched (See figure C.1).

```
Bufx = 0x320;
Offset = 0x400; /* 1 kB */
num_Blks = ceil((T_len + Offset)/nomBlk_len);
/* num_Blks = 247 */
Blk_len[0] = nomBlk_len - Offset;
/* Blk_len[0] = 19kB */
```

There are two methods for the Originating Source to create STUs for sending to the Destination. All STUs should be the maximum STU size (4 kB) except when the STU length would overrun: a Destination buffer boundary, a buffer boundary, or a transfer length. Additionally, if a Originating Source engine does not have an efficient buffer-gather mechanism, STUs may be clipped by the Originating Source buffer boundary. This example assumes the more efficient Originating Source capable of sending parts of at least two buffers in a single STU. Figure C.2 shows the tiling that would occur when STUs are also clipped by Originating Source buffer boundaries.

X->Y (6-10)

DATA{length}{Flags, S_count, Bufx, Offset, B_num)

DATA{x'1000'}(x'80', x'4', x'320', x'400', x'0')
{4kB STU at Offset 1kB}

DATA{x'1000'}(x'00', x'3', x'320', x'1400', x'0')
{4kB STU at Offset 5kB}

DATA{x'1000'}(x'00', x'2', x'320', x'2400', x'0')
{4kB STU at Offset 9kB}

DATA{x'C00'}(x'00', x'1', x'320', x'3400', x'0')
{3kB STU at Offset 13kB}

DATA{x'1000'}(x'04', x'0', x'321', x'0000', x'0')
{4kB STU at Offset 0kB}

As can be seen in this example, the Originating Source does most of the work to align the Buffer regions in the Destination. Figure C.1 shows how the first Block fits into Y's Bufx regions. The Originating Source sends the

largest STU size until it must account for the buffer boundaries on the Destination.

The last DATA Operation contains the State_Requested flag which triggers device Y to update the Op_limit and acknowledge the last block received. Each of the DATA Operations carries a synchronization value in the T_len parameter which the State_Response will echo so device X can synchronize its Op_limit parameter with the number of Operations that have been sent.

Y->X (11)

State_Response (S_count: C-limitY, T_len: SyncX+4, B_num: x'0')

The State_Response contains an updated Op_limit value in S_count called C-limitY. The synchronization value is contained in T_len and a Block number acknowledging this block and all lower numbered Blocks.

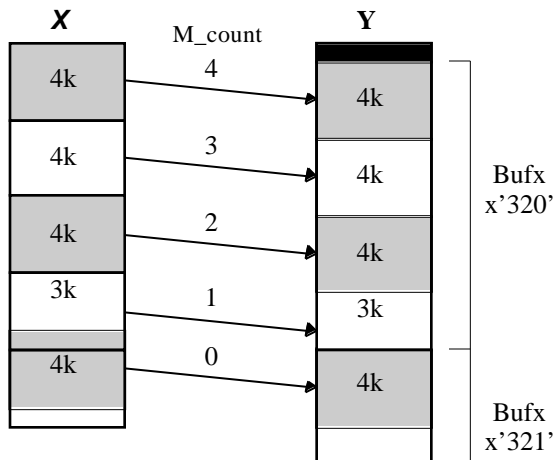


Figure C.1 – Buffer tiling

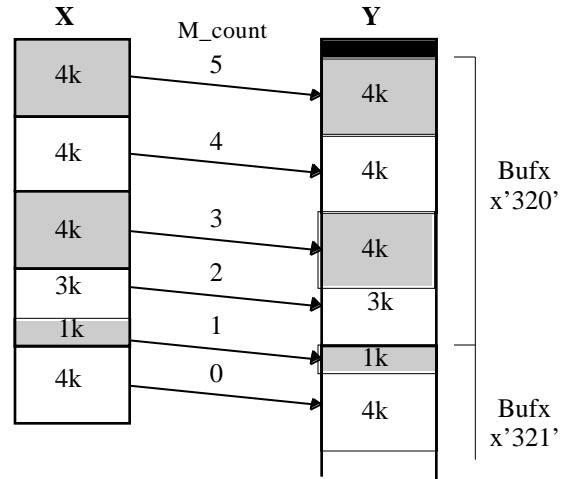


Figure C.2 – Alternate buffer tiling

C.4 Block 1 CTS

The following CTS sends an Offset and Bufx that should match where the Originating Source ended the last Block. Though this is the most likely case, the standard does not require the next Block's Bufx or Offset to sequentially follow the previous Block. Indeed the destination buffer pool may be reusing Bufx's that have already been used. The only requirement is that a Destination does not give a CTS for currently outstanding buffer regions.

Y->X (12)

CTS(Bufx: x'321', Offset: x'1000', B_num: x'1')

C.5 Block 246 transfer

The Scheduled Transfer is at the last Block.

Y->X (13)

CTS(Bufx: x'453', Offset: x'2000', B_num: x'F5')

The last Block is only 16.7k in length, but finishes off the transfer.

X->Y (14-18)

DATA{length}(Flags, S_count, Bufx, Offset, B_num)

DATA{x'1000'}(x'80', x'4', x'453', x'2000', x'F5')
{4kB STU at Offset 8kB}

DATA{x'1000'}(x'00', x'3', x'453', x'3000', x'F5')
{4kB STU at Offset 12kB}

DATA{x'1000'}(x'00', x'2', x'454', x'0000', x'F5')
{4kB STU at Offset 0kB}

DATA{x'1000'}(x'00' x'1', x'454', x'1000', x'F5')
{4kB STU at Offset 4kB}

DATA{x'02B8'}(x'04', x'0', x'454', x'2000', x'F5')
{696 byte STU at Offset 8kB}

The last DATA Operation requests a State_Response Operation. Though the State_Response in this example acknowledges the last Block, a Final Destination may require more time to finish buffering the Block and might only acknowledge the previous Block. In that case, device Y may wait and send a State_Request Operation and receive another State_Response acknowledging reception of the last Block.

Y->X (19)

State_Response (S_count: C-limitY, T_len:
SyncX+E49, B_num: x'F5')

C.6 Ending the Virtual Connection

Either side can end the Virtual Connection and free all resources associated with the Virtual Connection.

Y->X (20)

Port_Teardown(D_port: portX, S_port: portY,
Key: KeyX)

X->Y (21)

Port_Teardown_ACK(D_port: portY, S_port:
portX, Key: KeyY)

Y->X (22)

Port_Teardown_Complete(D_port: portX,
S_port: portY, Key: KeyX)

The Port_Teardown is a three-way handshake that decreases timeout dependency for releasing resources. The device sending the Port_Teardown_ACK can release all resources upon reception of the Port_Teardown_Complete with no worry about lost STUs in the network.